Norman F. Schneidewind, "Software Reliability Engineering for Client-Server Systems", Proceedings of the The Seventh International Symposium on Software Reliability Engineering, White Plains, New York October 30, 1996 - November 2, 1996, pp.226-235.

Abstract

Too often the assumption is made, when doing software reliability modeling and prediction, that the software involves either a single module or node. The reality in today's increasing use of multi node client-server systems is that there are multiple entities of software that execute on multiple nodes that must be modeled in a system context, if realistic reliability predictions and assessments are to be made. For example if there are N_c clients and N_s servers in a client-server system, it is not necessarily the case that a software failure in any of the N_c clients or N_s servers will cause the system to fail. Thus, if such a system were to be modeled as a single entity, the predicted reliability would be much lower than the true reliability because the prediction would not account for criticality and redundancy. The first factor accounts for the possibility that the survivability of some clients and servers will be more critical to continued system operation than others, while the second factor accounts for the possibility of using redundant nodes to allow for system recovery should a critical node fail. To address this problem, we must identify which nodes -- clients and servers -- are critical and which are not critical, as defined by whether these nodes are used for critical or non-critical functions, respectively.

1. Introduction

Popular software reliability models treat software as a single entity and model the failure process in accordance with this perspective. However in a distributed system, with multiple clients and servers, this approach is not applicable. We developed a software reliability model that takes into account the fact that not all software defects and failures result in *system failures* in a client-server system. In this model there are critical clients and servers: clients and servers with critical functions (e.g., network communication) that must be kept operational for the system to survive. There are also non-critical clients and servers with non-critical functions (e.g., email). These clients and servers also act as backups for critical clients and servers, respectively. The system does not fail unless all non-critical clients fail and one or more critical clients fail, or all non-critical servers fail and one or more critical servers fail. Our motivation was twofold: 1) Our literature search on related research yielded few articles on software reliability prediction that dealt with client-server systems, so there is the need to develop reliability models that address this environment; 2) Our work for the Marine Corps Tactical Systems Support Activity (MCTSSA) required the development of such a model because this is the type of system that is developed by this agency, where valid predictions of software reliability are important for evaluating the reliability of systems that will be deployed in the field. In addition to the development of a prediction model, it was also important to develop an approach to specifying software reliability requirements for client-server systems. These requirements must be stated in terms that recognize the difference between critical

and non-critical functions and that a software defect leading to a software failure does not necessarily result in a system failure. Furthermore the prediction methodology and the approach for specifying software reliability requirements must be consistent.

First we state the scope of our research. Then we discuss the client-server reliability prediction problem and provide definitions that are applicable to client-server systems. This is followed by comments on the related research we were able to find on software and system reliability modeling of client-server systems. Next we describe the need for consistency between client-server system reliability specifications and prediction models. Then we formulate the model, which consists of two major components: 1) probability of system failure, given the occurrence of node failures caused by software defects and 2) predictions of *Time to Failure* for various types of node failures, including the type that leads to system failure. At this point we apply the model to the Marine Corps' LOGAIS client-server system, in which we compare predicted with actual results. We close with some conclusions about the feasibility of this model and directions for future research.

1.1 Scope

We consider only software defects and failures and system failures that are caused by software failures. We exclude hardware failures. Also our model only includes predictions of software reliability and predictions of system reliability that are based on predictions of software failures. We exclude predictions of hardware reliability. Interestingly, in the Marine Corps' LOGAIS system (a logistical system for support amphibious operations), which we used as a test case for our research, 4084 (88.3 percent)of the 4584 defects were attributed to software [HEI96].

2. Client-Server Software Reliability Prediction

Too often the assumption is made, when doing software reliability modeling and prediction, that the software involves a *single* node. The reality in today's increasing use of *multi* node client-server systems is that there are multiple entities of software that execute on multiple nodes that must be modeled in a *system* context, if realistic reliability predictions and assessments are to be made. For example if there are N_c clients and N_s servers in a client-server system, it is not necessarily the case that a software failure in any of the N_c clients or N_s servers , which causes the node to fail, will cause the *system* to fail. Thus, if such a system were to be modeled as a single entity, the predicted reliability would be much lower than the true reliability because the prediction would not account for *criticality* and *redundancy*. The first factor accounts for the possibility that the survivability of some clients and servers will be more critical to continued *system* operation than others, while the second factor accounts for the possibility of using redundant nodes to allow for system recovery should a critical node fail. To address this problem, we must identify which nodes -- clients and servers -- are critical and which are not critical. We use the following definitions:

Node: A hardware element on a network, generally a computer, that has a network interface card installed [NOV95].

Client: A node that makes requests of servers in a network or that uses resources available through the servers [NOV95].

Server: A node that provides some type of network service [NOV95]

Client-Server Computing: Intelligence, defined either as processing capability or available information, is distributed across multiple nodes. There can be various degrees of allocation of computing function between the client and server, from one extreme of an application running on the

client but with requests for data to the server to the other extreme of a server providing centralized processing (e.g., mail server) and sharing information with the clients [NOV95]. We use the terms c*lient-server computing* and *distributed system* synonymously.

Critical function: An application function that must operate for the duration of the mission, in accordance with its requirement, in order for the system to achieve its mission goal (e.g., the requirement states that a military field unit must be able to send messages to headquarters and receive messages from headquarters during the entire time that a military operation is being planned). This type of function operates in the *network mode*, which means that the application requires more than a single client to perform its function; thus client to server or client to client communication is required.

Non-critical function: An application function that does not have to operate for the duration of the mission in order for the system to achieve its mission goal (e.g., it is not necessary to perform word processing during the entire time that a military operation is being planned). Often this type of function operates in the *standalone mode*, which means that a single client performs the application function; thus client to server or client to client communication is not required, except for the possible initial downloading of a program from a file server or the printing of a job at a print server.

Critical clients and servers: Nodes with critical functions, as defined above. These nodes must be kept operational for the system to survive, either by incurring no failures or by reconfiguring non-critical nodes to operate as critical nodes.

Non-critical clients and servers: Nodes with non-critical functions, as defined above. These nodes also act as backups for the critical nodes, should the critical nodes fail.

Software Defect: *Any* undesirable deviation in the operation of the software from its intended operation, as stated in the software requirements. A software defect may be *apparent*

Software Failure: A defect in the software that *causes* a node (either a client or a server) in a client-server system to be unable to perform its required function within specified performance requirements (i.e., a node failure caused by a *software* defect).

Non-Persistent Failure: A software failure that does not recur after canceling the offending program and retrying it, or rebooting, or using another node. The occurrence of such a failure is frequently hardware dependent (e.g., occurs if memory is insufficient) or environment dependent (e.g., occurs only if a certain sequence of operations or functions has occurred).

Persistent Failure: A software failure that recurs even after canceling the offending program and retrying it, or rebooting, or using another node. The occurrence of such a failure is independent of hardware capacity and operating environment.

System Failure: The state of a client-server system, which has experienced one or more node failures, wherein there are insufficient numbers and types of nodes available for the system to perform its required functions within specified performance requirements.

3. Related Research

In our literature search we found only a few articles that were directly related to our research; most of the articles were only tangentially related. We mention the more relevant ones. Hecht and Hecht model the availability of a distributed system as a function of capability [HEC95]. This allows tradeoffs to be made between these two factors. They point out that distributed systems do not just operate in one of two states -- operational or failed --

as in conventional reliability or availability analysis. Instead, distributed systems can operate in partial service or degraded states. Our approach of distinguishing between critical and non-critical functions, clients, and servers is consistent with this view. Hariri and Mutlu have developed a two level approach to analyze the availability of distributed systems [HAR95]. At the user level the availability of tasks was modeled by using a graph approach; at the component level Markov models were developed to analyze component availabilities. In contrast, our primary need is to predict the times when software failures will occur and the times when an accumulation of these failures will cause the system to fail. Lee and Iyer analyzed faults in the Tandem GUARDIAN operating system that resulted in processor failures and invoked backup processes to take over [LEE95]. One of our objectives is similar in that we analyze the types of software defects that can cause a node failure and the types of node failures that can cause a system failure. Kumur, Hariri, and Raghavendra developed a model for estimating the probability of successful execution of a distributed program and the probability that all programs of a set run successfully [KUM86]. In contrast, our focus is on predicting reliability at the node and system levels in a client-server system.

4. Client-Server Software Reliability Specification

In addition to the importance of modeling the correct system configuration when making reliability predictions, it is equally important to state software reliability requirements for a client-server system that are meaningful for the actual operational mode of the system. Typically, reliability requirements are stated as .999..... What does this mean in operational terms? Technically, according to the IEEE Standard Glossary on Software Engineering Terminology [IEE90], it means there is a .999 "probability that an item will perform a required function under stated conditions for a stated period of time". What is the "item" in the context of a client-server system? The IEEE definition suggests a single entity. In the definition, "function" is singular, whereas in a client-server system there are multiple functions. How do we operationalize the requirement of .999? Does it mean that a client should be able to execute a given function 99.9 percent of the attempts? How critical is this function relative to other functions? How do we allocate .999 among the various clients and servers?

What is needed is a software reliability specification that addresses the following: 1) definition of critical and non-critical functions; 2) definition of what constitutes "success" and "failure" in executing the functions; 3) consequences of failure to execute these functions correctly; 4) sequence of function execution; and 5) elapsed time in which functions must be completed. With this type of specification in hand, we can map it to a client-server architecture with a definition of the software and node failure states that would cause a *system* failure.

5. Model Formulation

By defining *System Nodes*, *Node Failure Probabilities*, and *Failure States*, we can formulate the probability of system failure *given* that a node failure has occurred.

5.1 System Nodes

N_{cc}: Number of Critical Client nodes.
$N_{nc}(t)$: Number of Non-Critical Client nodes.
N_{cs}: Number of Critical Server nodes.
$N_{ns}(t)$: Number of Non-Critical Server nodes.

where the total number of nodes is
$N(t)=N_{cc}+N_{nc}(t)+N_{cs}+N_{ns}(t)$ (1)

As long as the system survives, N_{cc} and N_{cs} are constants because a failure of a critical node will result in a non-critical node replacing it, if there is a

non-critical node available. Because many software failures are *non-persistent* and are specific to particular nodes, it is assumed that a non-critical node will replace a critical node, if one is available, in an attempt to keep the system operational. On the other hand, if a *persistent* software failure has occurred, it could cause the replacement node to also fail. A change in software configuration may be necessary on the former non-critical node in order to run the failed critical node's software. If a critical node fails, the **system fails**, *if there are no non-critical nodes available* on which to run the failed critical node's software.

In contrast, $N_{nc}(t)$ and $N_{ns}(t)$ are decreasing functions of operating time because these nodes replace failed critical nodes, and are not themselves replaced, where $N_{nc}(0)$ is the number of non-critical clients and $N_{ns}(0)$ is the number of non-critical servers at the start of system operation, respectively. In addition, if a non-critical node fails, the function that had been operational on the failed node can be continued on another node of this type and the system can continue to operate in a degraded state. When either a non-critical node replaces a critical node or a non-critical node fails, $N_{nc}(t)$ or $N_{ns}(t)$ is decreased by one, as appropriate.

We assume that when a node fails, and the failure does not result in a system failure, the network can be automatically or manually reconfigured by the network operating system or the operations personnel, respectively.

5.2 Node Failure Probabilities

We must also account for the following node failure probabilities:

p_{cc}: probability of a software defect causing a critical client node to fail.

p_{nc}: probability of a software defect causing a non-critical client node to fail.

p_{cs}: probability of a software defect causing a critical server node to fail.

p_{ns}: probability of a software defect causing a non-critical server node to fail.

Thus *given a node failure*, we have the following function for the probability of system failure:

$$P_{sys}/\text{node fails} = f(N_{cc}, p_{cc}, N_{nc}, p_{nc}, N_{cs}, p_{cs}, N_{ns}, p_{ns}) \quad (2)$$

The four probabilities are estimated from data in a defect database (defect descriptions, defect classifications, and administrative information) as follows:

$p_{cc} = \Sigma_i f_{cc}(I)/D$, where $f_{cc}(I)$ is the critical client node failure count in interval I; (3)

$p_{nc} = \Sigma_i f_{nc}(I)/D$, where $f_{nc}(I)$ is the non-critical client node failure count in interval I; (4)

$p_{cs} = \Sigma_i f_{cs}(I)/D$, where $f_{cs}(I)$ is the critical server node failure count in interval I; (5)

$p_{ns} = \Sigma_i f_{ns}(I)/D$, where $f_{ns}(I)$ is the non-critical server node failure count in interval I; (6)

and the total defect count across all intervals is

$$D = \Sigma_i d(I), \quad (7)$$

where I is the identification of an interval of operating time of the software and d(I) is the total defect count in interval I.

In a specific application, Boolean expressions are used to search the defect database and extract the failure counts (e.g., $f_{cc}(I)$) that are used to compute equations (3)-(6). These expressions specify the

conditions that qualify a defect as a node failure (e.g., defect that is a General Protection Fault that affects network operations on a Windows-based system).

5.3 Failure States

At a given time t, the system can be in one of three failure states that pertains to the survivability of the system, as follows, in decreasing order of capability:

Degraded - Type 1: A software defect in a non-critical node causes the node to fail. As a result, the system operates in a degraded state, with one less non-critical node. No reconfiguration is necessary because the failed node is not replaced.

Degraded - Type 2: A software defect in a critical node causes the node to fail. As a result, the system operates in a degraded state, but one that is more severe than *Type 1*, because there would be both a temporary loss of one critical node during reconfiguration and a permanent loss of one non-critical node (i.e., one of the non-critical nodes takes over the function of the failed critical node). Under certain conditions -- see Table 1 -- this type of node failure can cause a system failure.

The current version of the model assumes that node failures are not recoverable on the node where the failure occurred, *during the* mission. The next version of the model will contain a repair function to account for the case where a node failure is repaired and the node is put back into operation during the mission.

System Failure: The system fails under the following conditions: 1) all non-critical clients fail **and** one or more critical clients fail, **or** 2) all non-critical servers fail **and** one or more critical servers fail. The reason for this failure event formulation is that, in the event of a failed critical node, a non-critical node can be substituted, possibly with a different software configuration. However, if all non-critical clients (servers) fail, and one or more critical clients (servers) fail, there would be no non-critical clients (servers) left to take over for the failed critical clients (servers). The failure states are summarized in Table 1.

5.4 System Failure Probability

The probability that **one or more** critical clients N_{cc} fail, given that the software fails, is:

$$P_{cc}=1-(1-p_{cc})^{Ncc} \quad (8)$$

The probability that **all** non-critical clients $N_{nc}(t)$ have failed by time t, given that the software fails, is:

$$P_{nc}(t)=(p_{nc})^{Nnc(t)} \quad (9)$$

The probability that **one or more** critical servers N_{cs} fail, given that the software fails, is:

$$P_{cs}=1-(1-p_{cs})^{Ncs} \quad (10)$$

The probability that **all** non-critical servers $N_{ns}(t)$ have failed by time t, given that the software fails, is:

$$P_{ns}(t)=(p_{ns})^{Nns(t)} \quad (11)$$

Equations (8) and (9) assume that client failures are independent. This is the case because a failure in one client's software would not cause a failure in another client's software. However it is possible that a failure in server software could cause a failure in client software, such as a client accessing a server that has corrupted data. Also, equations (10) and (11) assume that server failures are independent. This is the case because a failure in one server's software would not cause a failure in another server's software. However it is possible that a failure in client software could cause a failure in server software, such as a client with corrupted data accessing a server. We have found no case of client failures that were caused by server failures nor of the converse in the data we

have examined. Of course, this is not to suggest that these events could not happen in general. To account for the possibility of these events, we would need to include the conditional probability of a client failure, given a server failure, and the converse. This model formulation is beyond the scope of this paper and will appear in a future paper.

Combining (8), (9), (10), and (11), the probability of a system failure by time t, given that a node fails, is:

$$P_{sys}/\text{node fails}=[P_{cc}][P_{nc}(t)]+[P_{cs}][P_{ns}(t)]=$$

$$[1-(1-p_{cc})^{Ncc}][(p_{nc})^{Nnc(t)}]+[1-(1-p_{cs})^{Ncs}][(p_{ns})^{Nns(t)}] \quad (12)$$

and the probability of a node failure due to software is:

$$p_{sw}=p_{cc}+p_{nc}+p_{cs}+p_{ns} \quad (13)$$

5.5 Model Concepts

The model concepts are illustrated in Figures 1 and 2, where there are five critical clients, five non-critical clients, one critical server, and one non-critical server. Figure 1 shows a surviving configuration, where a critical client fails and a critical server fails but there are non-critical clients and a non-critical server to take over the functions of the failed nodes. The consequence of this configuration is a *Degraded - Type 2* failure state. Figure 2 shows a failing configuration where there are no non-critical clients and server to take over for the critical failing nodes. The consequence of this configuration is a *system failure*. In both figures, for illustrative purposes, we show both a failed critical client and a failed critical server. A more typical case is when only one of the critical nodes fails at a time.

5.6 Time to Failure Prediction

In order to make *Time to Failure* predictions for each of the four types of node failures, we first analyze the defect data to determine what type of software defects could cause each of the four types of node failures; then we partition the defect data accordingly. More will be said about this process when we apply the model. Next we apply equation (14) of the *Schneidewind Software Reliability Model* [AIA93, KEL95, LYU 96, SCH92, SCH93] to make each of the four predictions, using the *SMERFS* software reliability tool [FAR93]. In equation (14), $T_f(t)$ is the predicted time (intervals) until the next F_t failures (one or more) occur, á and â are failure rate parameters, s is the first interval where the observed failure data is used, t is the current interval, and $X_{s,t}$ is the cumulative number of failures observed in the range s,t.

$$T_F(t)\text{ '}[(\log[á/(á\&â(X_{s,t}\% \quad \text{for } (á/â)>(X_{s,t}\%F_t) \quad (14)$$

Time to Failure predictions are made for critical clients, non-critical clients, critical servers, and non-critical servers. As the predicted failure times are recorded, we observe whether the condition for system failure, as defined previously, has been met. If this is the case, a predicted system failure is recorded. Thus, in addition to monitoring the types of predicted failures (e.g., critical client), the process also involves monitoring $N_{nc}(t)$ and $N_{ns}(t)$ to identify the time t when either is reduced to zero, signifying that the supply of non-critical clients or non-critical servers has been exhausted. In this situation, a failure of a critical client or critical server, respectively, will result in a system failure. Thus we predict a system failure when the following expression is true:

$$((\text{Predict critical client failure})\vee(N_{nc}(t)=0))\wedge((\text{Predict critical server failure})\vee(N_{ns}(t)=0)) \quad (15).$$

If our predictions produce multiple node failures in the same interval (e.g., critical client and critical server), we record multiple failures for that interval.

6. Application of the Model

6.1 Analysis of the Defect and Failure Data

Now we apply the software reliability modeling approach that has been described. We apply the model to the Marine Corps LOGAIS system -- a client-server logistical support system. In this system it is important that the reliability specification distinguish between failure states *Degraded-Type 1*, *Degraded-Type 2*, and *System Failure*, as previously defined (i.e., distinguish between node failures that cause performance degradation but allow the system to survive, and node failures that cause a system failure). We make this distinction when analyzing the system's defect data. The defect data used in the example are from the LOGAIS defect database management system [MHB96, MTP96]. We use the network configurations in Figures 1 and 2.

In this Windows-based client-server system, we use the types of clients and servers previously defined, with corresponding types of defects and failures, as identified in the defect database [MHB96, MTP96], and the following short-hand notation for identifying the attributes of the defect database:

o S: Software Defect
o G: General Protection Fault (GPF)
o N: Network Mode Failure
o C: System Crash

The LOGAIS defect database was queried in order to identify the software defects that qualify as node failures. The following Boolean expressions, corresponding to the four types of node failures, were used:

1. Critical Client Failure: COUNT as failures WHERE (S∨G∨N∨*not*C). A *GPF* causes a node failure (*Degraded-Type 2*) on a critical client, a client which must maintain communication with other nodes on the network (*Network Mode*), and the failure does not cause a *System Crash* (loss of server).

2. Non-Critical Client Failure: COUNT as failures WHERE (S∨G∨*not*N∨*not*C). A *GPF* causes a node failure (*Degraded-Type 1*)on a non-critical client, a client which does not have to maintain communication with other nodes on the network (*Standalone Mode*), and the failure does not cause a *System Crash* (loss of server).

3. Critical Server Failure: COUNT as failures WHERE (S∨*not*G∨N∨C). A *System Crash* causes a node failure (*Degraded-Type 2*)on a critical server, a server which must maintain communication with other nodes on the network (*Network Mode*), and the failure is not a *GPF;* it is more serious, resulting in the loss of a server.

4. Non-Critical Server Failure: COUNT as failures WHERE (S∨*not*G∨*not*N∨C). A *System Crash* causes a node failure (*Degraded-Type 1*)on a non-critical server, a server which does not have to maintain communication with other nodes on the network, and the failure is not a *GPF;* it is more serious, resulting in the loss of a server.

The above classification associates *GPF* with clients and *System Crash* with servers; it also associates *Network Mode Failures* with critical node failures. Note that this is only an example. For other systems, different defect and failure classifications may be appropriate.

Taking the union of 1-4 we form the total failure count:

5. Total Failure Count: COUNT as failures WHERE (S∨((G∨*not*C)∧(*not*G∨C))). This expression is used to verify the correctness of expressions 1-4 because it should equal their sum.

Upon querying the defect database, using the above expressions, we arrived at the failure counts listed in Table 2 in the range 51,61. This is a sample of the node failure database, selected because it is the prediction range for some of the comparisons that will be made later between predicted and actual *Time to Failure*. The failure counts corresponding to types 1-5, above are summarized in Table 3, which shows

the empirical probabilities of node failure, where we used equations (3)--(7) and (13) to compute them.

6.2 Application Predictions

6.2.1 Time to Failure

Using equation (14) and failure data in the observed range 1-50 (not shown) , we made predictions for *Time to Failure*, for t>50 days, for critical clients, non-critical clients, and non-critical servers, in Tables 4, 5 and 6, respectively. The predictions are made for a given numbers of failures (time to one failure for t>50 days, time to two failures for t>50 days, etc.). The predictions are compared with the actual failure data, with the relative error and average relative error for cumulative values shown. In the case of critical servers, there are only two actual failures, both of which occur in the observed range. Only one prediction of *Time to Failure* for **one more failure** could be made at t=50 for critical servers because the predicted remaining failures at t=50 is 1.40 ; therefore, critical server failures are not tabulated. In the case of non-critical nodes, the failure data is sufficiently dense to allow a failure count interval of one day. In the case of critical clients, the failure data was sparse; thus a five day interval was used for prediction, with these predictions converted to the one day intervals shown in Table 4. We note that predictions are difficult to make with this type of data because the defects and failures are not recorded in CPU execution time. Rather they are recorded in calendar time in batches, as shown in the Table 2, based on administrative convenience. Many of these batches are submitted at the end of a workday. This time becomes the "submit date".

Using the data in Tables 4-6, we merge and sequence the various types of failure predictions in Table 7. The purpose of this table is to construct the scenario of failures and surviving non-critical nodes so that the time of *System Failure* can be predicted. The table shows that seven node failures (i.e., the sequence NS, NC, NC, CC, NC, NC, CC) are predicted to occur before the system is predicted to fail. This occurs at t=61.07 days when there are no non-critical clients available and a critical client fails. No critical server failures are shown in this table because the prediction of *Time to Failure* of 99.35 days cumulative is beyond the prediction range of interest in this example.

Using the data in Table 7, we plot predicted cumulative failures and number of available non-critical clients versus cumulative time to failure in Figure 3. This graph shows the accumulation of node failures, with the corresponding reduction in the available non-critical clients, until the maximum allowable failures occurs, and the system fails.

Using the data in Tables 4-6, we merge and sequence the various types of actual failures in Table 8. Similar to Table 7, the purpose of this table is to construct the scenario of actual failures and surviving non-critical nodes so that the actual time of *System Failure* can be determined and compared with the predicted values. As in the case of the predictions, this table shows that seven node failures (i.e., the sequence NC, NS, NC, NC, NC, NC, CC) occur before the system fails. This occurs at t=61days when there are no non-critical clients available and a critical client fails. No critical server failures are shown in this table because they occurred prior to the range of this example.

Using the data in Table 8, we plot actual cumulative failures and number of available non-critical clients versus cumulative time to failure in Figure 4. The shape of Figure 4 is caused by multiple failures occurring on the same day in some cases. In comparing Figures 3 and 4, we see that in each case seven node failures are required to cause a system failure and that the system fails on Day 61; however, the supply of non-critical clients becomes exhausted earlier in the actual case.

6.2.2 Probability of System Failure

Lastly, using equation (12), we predict the probability of system failure, given a node failure, in column 5 of Table 9, as the system progresses through the predicted failure scenario that was shown in Table 7 and Figure 3. Except for row 2 in Table 9, the actual probability is the same as the predicted probability because the actual failure scenario that was shown in Table 8 and Figure 4 produces the same numbers of non-critical clients and servers that are shown in columns 6 and 7, respectively. Because the predicted and actual failure scenarios are identical, except for row 2, the predicted time to failure and type of node failure, columns 1 and 2, respectively, can be compared in with the corresponding actual values in columns 3 and 4, for given probabilities of system failure. These values were reproduced from Tables 7 and 8, respectively. Because for a given P_{sys}*/node fails*, the cumulative time to failure occurs later for the predicted values, the model is a bit optimistic with respect to reality for this example. Note that the in the last row of Table 9 the system has not yet failed. This occurs when a critical client fails at Day 61.07 *predicted* (see Table 7 and Figure 3) and at Day 61 *actual* (see Table 8 and Figure 4). At this time there are no non-critical clients left to replace the failed critical client.

The significant results that emerge from this analysis are that: 1) The P_{sys}*/node fails* is only significant (.029790) when the supply of both non-critical clients and non-critical servers has been exhausted and 2) P_{sys}*/node fails* is significantly lower than the probability of *any* type of node failure caused by a software defect: p_{sw}=.065705, obtained from equation (13) and computed in Table 3. Thus evaluations of system reliability should recognize that *software failures are not necessarily equivalent to system failures* and that assessments of software reliability that treat every failure as equivalent to a system failure will grossly understate system reliability.

7. Conclusions and Future Research

Based on the above approach, it appears feasible to develop a system software reliability model for a client-server system. In order to implement the approach, it is necessary to partition the defects and failures into classes that are then associated with critical and non-critical clients and servers. Once this is done, predictions are made of *Time to Failure* for each type; the predictions are classified according to those that would result in a node failure caused by a software defect and those that would result in a system failure caused by a series of software defects. Then the probability of system failure is computed. A significant result of the research is that software failures should not be treated as the equivalent of system failures because to do so would grossly understate system reliability.

In future research we will deal with the problem of how to apply the model to a system that has a large number of nodes. The technique that we described for monitoring the times when predicted node and system failures occur would be cumbersome for a large system. It appears that a program must be written to automate this process. Other possible future research activities include the following: extend the model to include hardware failures; develop measures of performance degradation, as nodes fail; include a node repair rate to reflect the possibility of recovering failed nodes during the operation of the system; apply smoothing techniques, such as the moving average, to mitigate anomalies in calendar time defect data.

8. References

[AIA93] Recommended Practice for Software Reliability, R-013-1992, American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[FAR93] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

[HAR95] Salim Hariri and Hasan Mutlu, "Hierarchical Modeling of Availability in Distributed Systems', IEEE Transactions on Software Engineering, Vol. 21, No. 1, January 1995, /pp. 50-56.

[HEC95] Herbert Hecht and Myron Hecht, "Dependability Assessment for Decentralized Systems", Proceedings of the International Symposium on Autonomous Decentralized Systems, Phoenix, AZ. March 1995, 9 pages.

[HEI96] Judie Heineman, Norman Schneidewind, and Kenneth Warburton, "Software Reliability Engineering Process Experience Report", Proceedings of The Eighth Annual Software Technology Conference, Salt Lake City, Utah, 21-26 April 1996, 17 pages.

[IEE90] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12.1990.

[KEL95] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", Proceedings of the AIAA Computing in Aerospace 10, San Antonio, TX, March 28, 1995, pp. 1-8.

[KUM86] V.K. Prasanna Kumar, Salim Hariri, and C.S. Raghavendra, "Distributed Program Reliability Analysis", IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, January 1996, pp. 42-50.

[LEE95] Inhwan Lee and Ravishankar.K. Iyer, "Software Dependability in the Tandem GUARDIAN System", IEEE Transactions on Software Engineering, Vol. 21, No. 5, May 1995, pp. 455-467.

[LYU96] Michael R. Lyu (Editor-in-Chief), Handbook of Software Reliability Engineering, Computer Society Press, Los Alamitos, CA and McGraw-Hill, New York, NY, 1995.

[MHB96] MCTSSA Software Reliability Handbook, Norman F. Schneidewind and Judie A. Heineman, Naval Postgraduate School, January 10, 1996.

[MTP96] MCTSSA Software Reliability Engineering Training Plan, Norman F. Schneidewind and Judie A. Heineman, Naval Postgraduate School, January 10, 1996.

[NOV95] Werner Feibel, Novell's Complete Encyclopedia of Networking, Novell Press, San Jose, CA, 1995.

[SCH93] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", IEEE Transactions on Software Engineering, Vol. 19, No. 11, November 1993, pp. 1095-1104.

[SCH92] Norman F. Schneidewind and T. W. Keller, "Application of Reliability Models to the Space Shuttle", IEEE Software, Vol. 9, No. 4, July 1992 pp. 28-33.

Table 1
Failure States

	Degraded - Type 1	Degraded - Type 2	System Failure
Non-Critical Client	Node Failure	Does Not Apply	Does Not Apply
Critical Client	Does Not Apply	Node Failure(s) and $N_{nc}(t)>0$	Node Failure(s) and $N_{nc}(t)=0$
Non-Critical Server	Node Failure	Does Not Apply	Does Not Apply
Critical Server	Does Not Apply	Node Failure(s) and $N_{ns}(t)>0$	Node Failure(s) and $N_{ns}(t)=0$

Table 2
Chronological Node Failure Count Database (Sample)

CC: Critical Client Node Failure
NC: Non-Critical Client Node Failure
CS: Critical Server Node Failure
NS: Non-Critical Server Node Failure

Interval	Defect ID	Number	Submit Date	CC	NC	CS	NS
51	2633,2634	2	1/24/95		X		
51	2635,2636,2637,2638	4	1/24/95				X
52							
53	2661,2662,2663,2664	4	1/26/95		X		
54	2641,2644,2645,2669, 2671,2672,2673,3003	8	1/27/95		X		
54	2640,2643,2670,2674, 2675,2676,2783	7	1/27/95				X
55	2450	1	1/30/95		X		
56							
57							
58	2487	1	2/2/95				X
59	2511,2512,2513	3	2/3/95		X		

60							
61	3025,3026,3027,3029	4	2/7/95	X			

Table 3
Summary of Node Failures (4048 Software Defects)

	Number of Failures	Probability
1. Critical Client	24	p_{cc}=.005929
2.Non-Critical Client	83	p_{nc}=.020250
3.Critical Server	2	p_{cs}=.000494
4. Non-Critical Server	158	p_{ns}=.039032
5. Total	267	p_{sw}=.065705

Table 4
Critical Client Predictions Made at Time=50 Days
Observed Range=1,50 Days; Failure Count=11; Prediction Range>50 Days

Predicted			**Actual**		
Given Number of Failures	Time to Failure (Days)	Cumulative Time to Failure (Days)	Time to Failure (Days)	Cumulative Time to Failure (Days)	Relative Error (Percent)
1	5.19	55.19	11	61	-9.52
2	11.07	61.07	11	61	+.11
3	17.88	67.88	11	61	+11.28
4	25.95	75.95	11	61	+24.51
5	35.86	85.86	36	86	-.16

Average=9.12%

Table 5
Non-Critical Client Predictions Made at Time=50 Days
Observed Range=1,50 Days; Failure Count=36; Prediction Range>50 Days

Predicted	**Actual**	

Given Number of Failures	Time to Failure (Days)	Cumulative Time to Failure (Days)	Time to Failure (Days)	Cumulative Time to Failure (Days)	Relative Error (Percent)
1	2.41	52.41	1	51	+2.76
2	4.87	54.87	1	51	+7.59
3	7.37	57.37	3	53	+8.25
4	9.92	59.92	3	53	+13.06
5	12.52	62.52	3	53	+17.96

Average=9.92%

Table 6
Non-Critical Server Predictions Made at Time=50 Days
Observed Range=1,50 Days; Failure Count=108; Prediction Range>50 Days

Predicted			**Actual**		
Given Number of Failures	Time to Failure (Days)	Cumulative Time to Failure (Days)	Time to Failure (Days)	Cumulative Time to Failure (Days)	Relative Error (Percent)
1	1.96	51.96	1	51	+1.88
2	3.93	53.93	1	51	+5.75
3	5.90	55.90	1	51	+9.61
4	7.87	57.87	1	51	+13.47
5	9.84	59.84	4	54	+10.81

Average=8.30%

Table 7

Predicted Time to Failure When Failures are Merged and Sequenced. Observed Range=1,50 Days; Prediction Range=51,61 Days

CC: Critical Client NC: Non-Critical Client NS: Non-Critical Server

Cumulative Time to Failure (Days)	Time to Failure (Days)	Type of Failure	Number of Non-Critical Clients Available	Number of Non-Critical Servers Available
50	1.96		5	1
51.96	.45	NS	5	0
52.41	2.46	NC	4	0
54.87	.32	NC	3	0
55.19	2.18	CC	2	0
57.37	2.55	NC	1	0
59.92	1.15	NC	0	0
61.07		CC	**System Failure**	

Table 8

Actual Time to Failure When Failures are Merged and Sequenced. Range=51,61 Days

CC: Critical Client NC: Non-Critical Client NS: Non-Critical Server

Cumulative Time to Failure (Days)	Time to Failure (Days)	Type of Failure	Number of Non-Critical Clients Available	Number of Non-Critical Servers Available
50	1		5	1
51	0	NC,NS	4	0
51	2	NC	3	0
53	0	NC	2	0
53	0	NC	1	0
53	8	NC	0	0
61		CC	**System Failure**	

Table 9

Probability of System Failure

Predicted Cumulative Time to Failure (Days)	Predicted Type of Node Failure	Actual Cumulative Time to Failure (Days)	Actual Type of Node Failure	Probability of System Failure Given a Node Failure	Number of Non-Critical Clients Available	Number of Non-Critical Servers Available
50		50		0.000019	5	1
51.96	NS			0.000494*	5*	0*
52.41	NC	51	NC,NS	0.000494	4	0
54.87	NC	51	NC	0.000494	3	0
55.19	CC	53	NC	0.000506	2	0
57.37	NC	53	NC	0.001087	1	0
59.92	NC	53	NC	0.029790	0	0

*** Applies only to predicted values.**

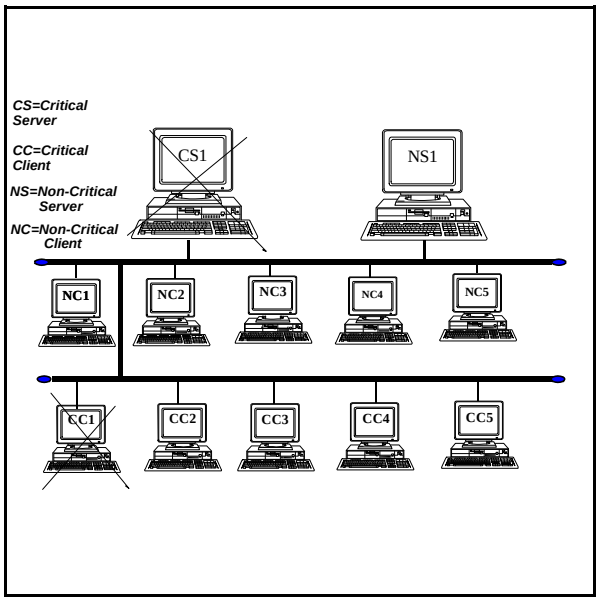

Figure 1. Surviving Configuration

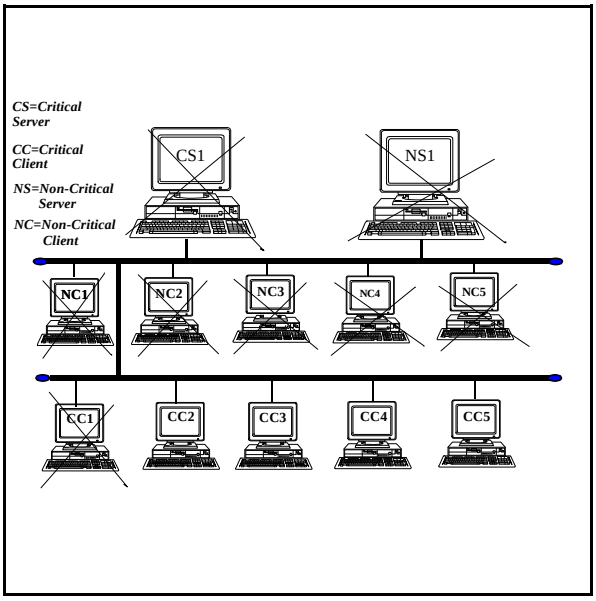

Figure 2. Failing Configuration

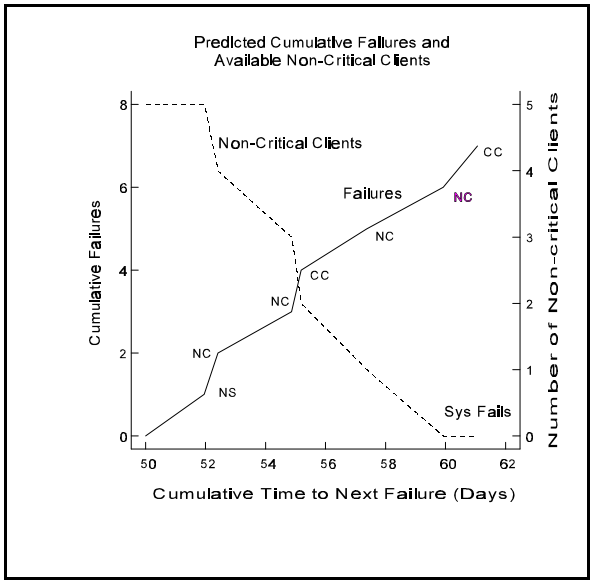

Figure 3. Predicted Node and System Failures

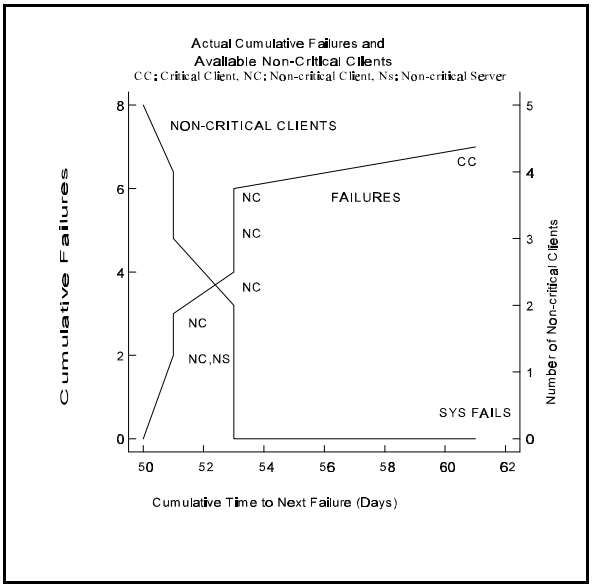

Figure 4. Actual Node and System Failures